Exceptional service in the national interest

Sandia National Laboratories

In situ Visualization with the Sierra Simulation Framework Using ParaView Catalyst

Jeff Mauldin, Thomas Otahal, David Karelitz,
Alan Scott, Warren Hunt, Nathan Fabian

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

What is *In situ* Visualization?

Sandia National Laboratories

- A typical visualization workflow:
  - Run simulation and output the full 3d mesh data at some sparse time interval.
  - After simulation completes, load output data into a visualization tool such as ParaView, EnSight, VisIt, etc.
- *In situ* Visualization executes a visualization pipeline on the processors running the simulation during or between simulation time steps.
- *In transit* Visualization executes a visualization pipeline on a set of compute nodes separate from simulation nodes; sending data across an HPC network.
- Data products other than images are possible such as: decimated isosurfaces, mesh subsets, plots, and analysis results.

In situ and *in transit* workflows

Sandia National Laboratories

- *In situ* processing provides "tightly-coupled" analysis capabilities through libraries linked directly with the simulation. SNL has collaborated on developing an *in situ* capability designed for this purpose.

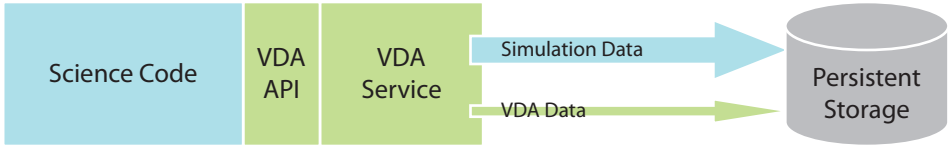


Diagram of *in situ* workflow, accomplished through the use of *Catalyst*, an open source, ParaView-based analysis library.

- *In transit* processing provides "loosely-coupled" analysis capabilities by performing the analysis on separate processing resources. SNL provides this capability through a "data services" capability designed for this purpose.

Diagram of *in transit* workflow, in which the simulation code communicates with data service nodes to perform analysis operations.

What is Catalyst and how is it used?

Sandia National Laboratories

- http://catalyst.paraview.org (Managed by Kitware Inc.)
- The Catalyst library links with the simulation code and enables *in situ* analysis and visualization.
  - Language support for Python, C++, and Fortran.
- Catalyst editions package various levels of functionality such as rendering, Python support, and various classes of VTK filters.
- Catalyst Requirements:
  - Write adapter code to create a VTK mesh from the simulation data either copying data or using the simulation's data structures.
  - Simulation code calls ParaView Catalyst with the created VTK mesh each time in situ output is required.
  - Create a Python visualization script from the ParaView co-processing plugin, or drive in situ visualization through direct C++ implementation.

The Sierra Simulation Framework

Sandia National Laboratories

- Sandia simulation code framework for massively parallel finite element simulations.
- Physics for solid mechanics, fluid dynamics, heat transfer, etc.
- Simulation codes share a common input deck syntax and parsing engine, which allows codes to be coupled easily from the level of the input deck.
- Sierra's I/O subsystem writes out full-mesh, unstructured grid Exodus II files with associated data at a user controlled time interval.

Initial Attempt at Integration with Sierra

Sandia National Laboratories

- Catalyst is statically linked to Sierra. Large binary code size increase for users not using *in situ* visualization.
- Minimal alteration to Sierra input deck—basically alternative output section and python file to call—all flexibility arbitrarily in Python code.
- Requires analysts to create a script with the Catalyst ParaView plugin (which may need adjusting), potentially do Python programming or employ help of a visualization programmer, and then have Python scripts with somewhat arbitrary ways to change parameters.
- Not easily testable.
- High barrier to first use by analysts already familiar with Sierra.
- Does not really "feel" integrated.
- No support from simulation input checking. Simulation has to start running and call back visualization script before any error checking occurs.

Current Integration Strategy

Sandia National Laboratories

- Command block structure and command syntax looks like already existing Sierra commands.
- Visualization commands are available in the existing Sierra GUI editor tools the same as other Sierra commands.
- Minimal changes are required to an existing Exodus II output command blocks to produce a visualization (convention over configuration).
- Increasingly advanced visualization capabilities available with additional commands in input deck, always trying to provide “good defaults” to let user create visualization without having to learn a bunch of details first.
- Complete ParaView capability still available to users via already existing python scripting technique.
- Catalyst linked to Sierra as a dynamic library at run-time. Sierra based codes that do not require *in situ* visualization do not load the dynamic library.

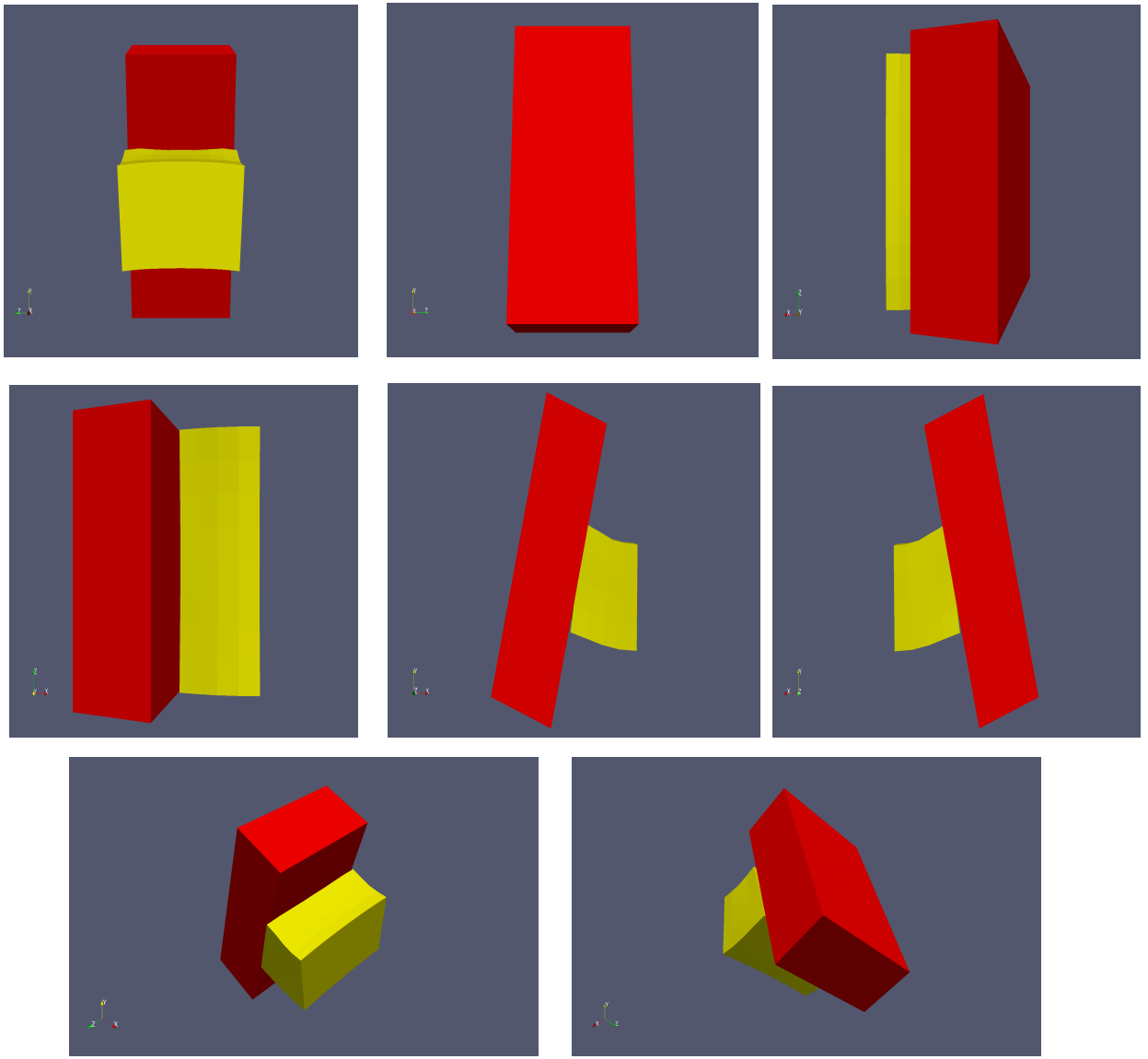
Minimal Example

- Change the type of output database to catalyst from exodus.

```
# catalyst block, accepting all default values
begin results output catalyst_example_results_block
  At Step 0, Increment = 200
  # export displacement
  # catalyst will displace geometry by default
  nodal Variables = displacement as displ
  database type = catalyst
end results output catalyst_example_output_block
```

Default Camera Views

Default behavior is to provide 8 views of mesh, 6 axis-aligned and 2 from first and eight octant. Default coloring is by block id. View is centered on data bounding box. Works well for meshes with interesting geometry, not as well for simulations where anything interesting is happening inside the mesh. Requires minimal changes to the input deck.

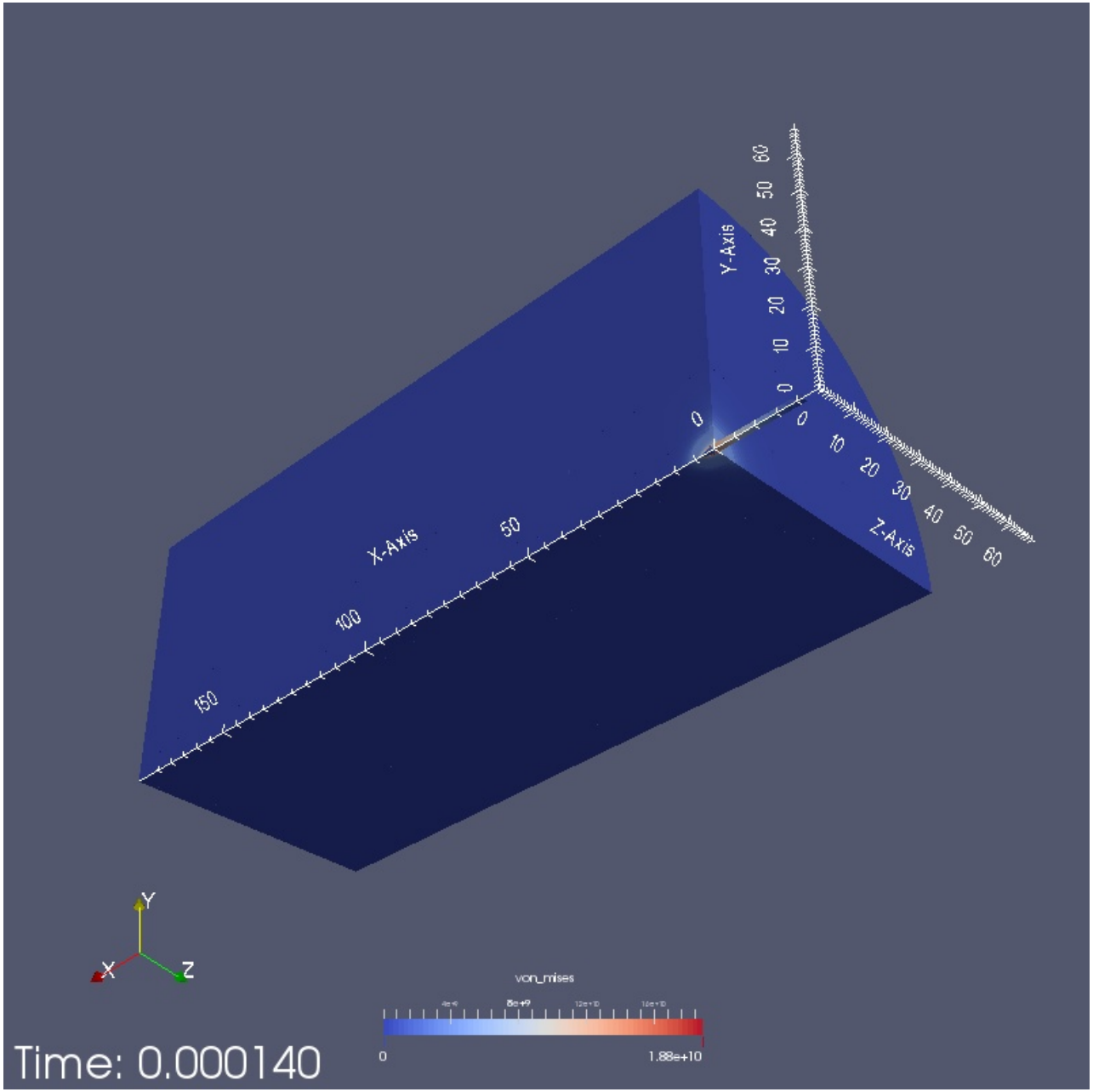


Catalyst Block Example

Sandia National Laboratories

- Adding a catalyst block allows for additional configuration of default parameters. Here, we color the mesh by von Mises stress.

```
# catalyst block, accepting all default values
begin results output catalyst_example_results_block
  At Step 0, Increment = 200
  # export displacement
  # catalyst will displace geometry by default
  nodal Variables = displacement as displ
  database type = catalyst
  element Variables = von_mises

  begin catalyst
    show axes = true
    show time annotation = true
    color by scalar = von_mises
  end
end results output catalyst_example_output_block
```



Sandia National Laboratories

Image Set and Camera Blocks

Sandia National Laboratories

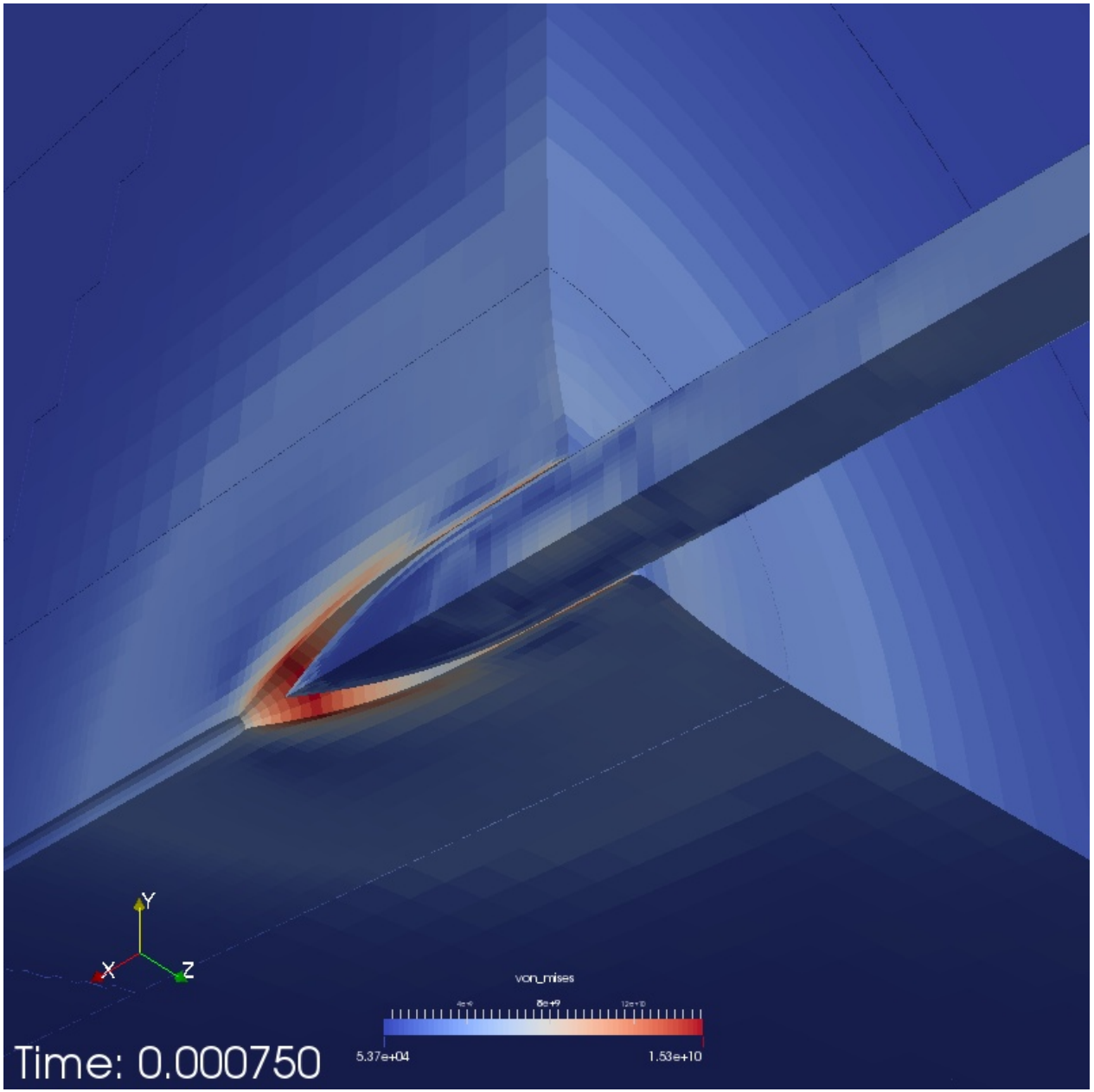
- More complexity to allow multiple image sets (e.g. colored by different variables).
- Extensive camera controls.
  - Look at focal point based on absolute position or position relative to bounding box data.
  - Look at element or node (following).
  - Distance between camera and focal point in absolute terms or relative to data bounding box size.
  - Specify look direction(s).
  - Alternatively specify camera position and look direction.
  - Easy to bring in camera settings from other packages, although currently tedious.

Image Set and Camera Block Example (Time annotation)

Sandia National Laboratories

```
…
  element Variables = von_mises
  nodal Variables = velocity

  begin catalyst
    begin camera myZoomCamera
      look at absolute point = 0 0 0
      look direction = 1 1 1
      look at relative distance = 0.5
    end
    begin imageset vm_set
      camera = myZoomCamera
      color by scalar = von_mises
      show time annotation = true
    end
    begin imageset vel_set
      camera = myZoomCamera
      color by vector magnitude = velocity
    end
  end
…
```

Sandia National Laboratories



Additional Features and Block Types

Sandia National Laboratories

- Representation blocks support color legend management, background and text colors, time annotation, and surface/edge/wireframe display of the geometry.
- Operation blocks provide chains of visualization operations. Operation block types for clips, slices, isosurfaces, and thresholds.
- Plot over time block.
- Scatter plot block.

Memory Usage

Sandia National Laboratories

- Virtual memory image of code on nodes is significant. Python wrapping of ParaView is large piece of this.
  - With shared libraries and multiple cores per node this cost is amortized.
  - C++ implementation would reduce code size greatly.
  - Exploring use of Catalyst editions containing fewer libraries.
- Mesh data is copied into VTK datasets
  - New shallow copy capability can limit this issue.
- Pipeline operations have the potential to use more memory.
- Discovered multi-image RenderView issue. Resolved by sharing RenderView among image pipelines.

Catalyst Resource Usage Test

Sandia National Laboratories

Simulation problem details:

Number of nodes = 1123420
Number of elements = 1050604
Element type = hex 8

Sierra Adagio thermal stress problem.

Total number of Sierra iterations = 23942

Number of calls to Catalyst/Exodus = 27

HPC hardware details:

8 cores per compute node
CPU: 2.93 GHz dual socket/quad core, Nehalem X5570 processors
Compute Memory: 12 GB RAM per compute node (1.5 GB per core)
Interconnect: 3D torus InfiniBand

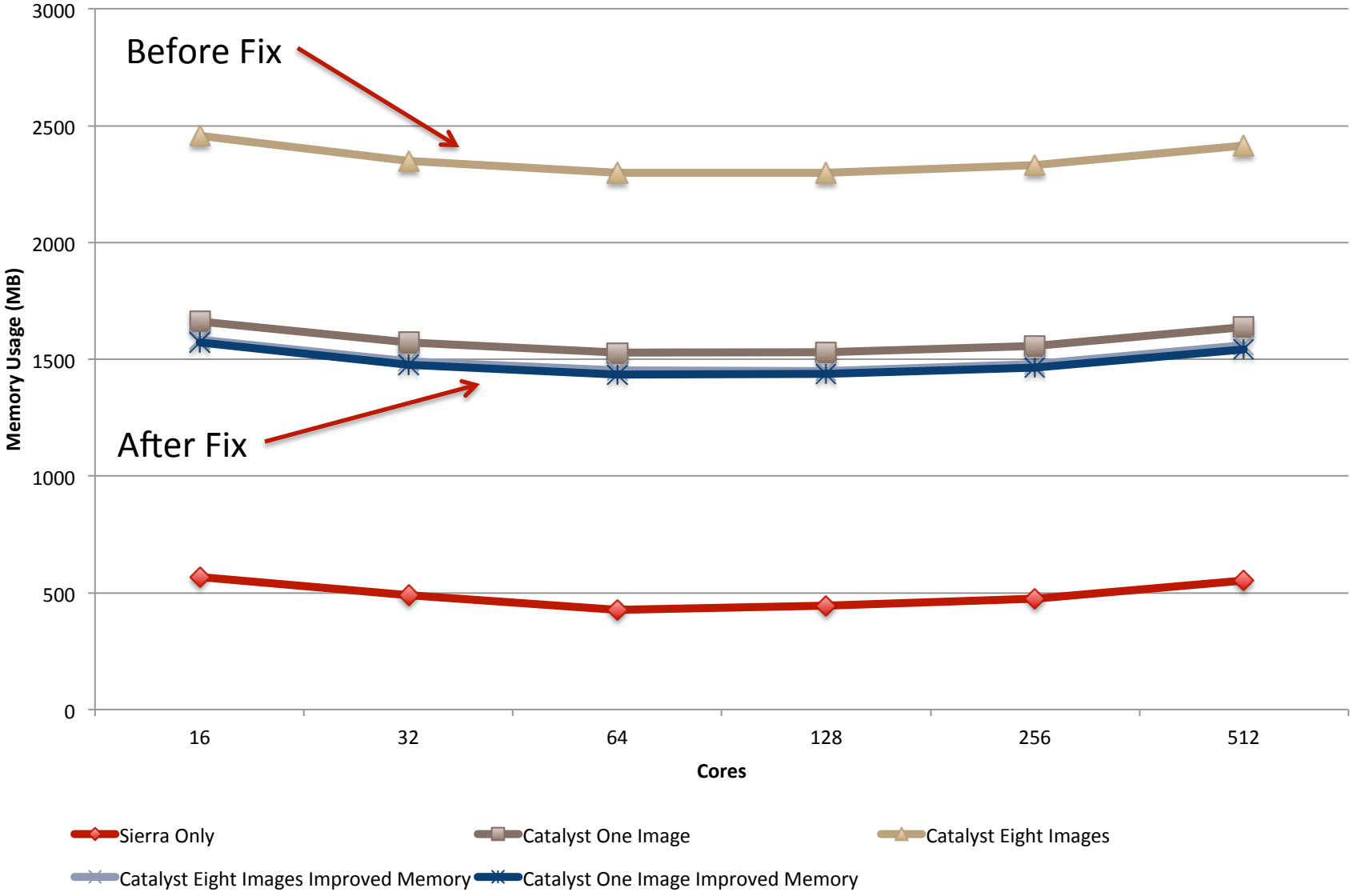
Catalyst Resource Usage Test

Sandia National Laboratories

```
begin catalyst

  begin threshold threshold_vm
    variable scalar = VON_MISES
    keep between = 5e3 3.54e4
  end threshold

  begin clip clip_in_half
    input = threshold_vm
    absolute point on plane = 0.0 0.1 0.2
    plane normal = -1 0 0
  end clip

  begin camera my_camera
    camera at absolute point = 1.586 0.724 -0.324
    look at absolute point = 0.0410 -0.027 0.182
  end camera

  begin imageset show_images
    operation = clip_in_half
    camera = my_camera
    color by scalar = VON_MISES
    color legend range = 1.41 3.54e4
  end imageset

  begin plot over time
    variable scalar = VON_MISES
  end plot over time

end catalyst
```

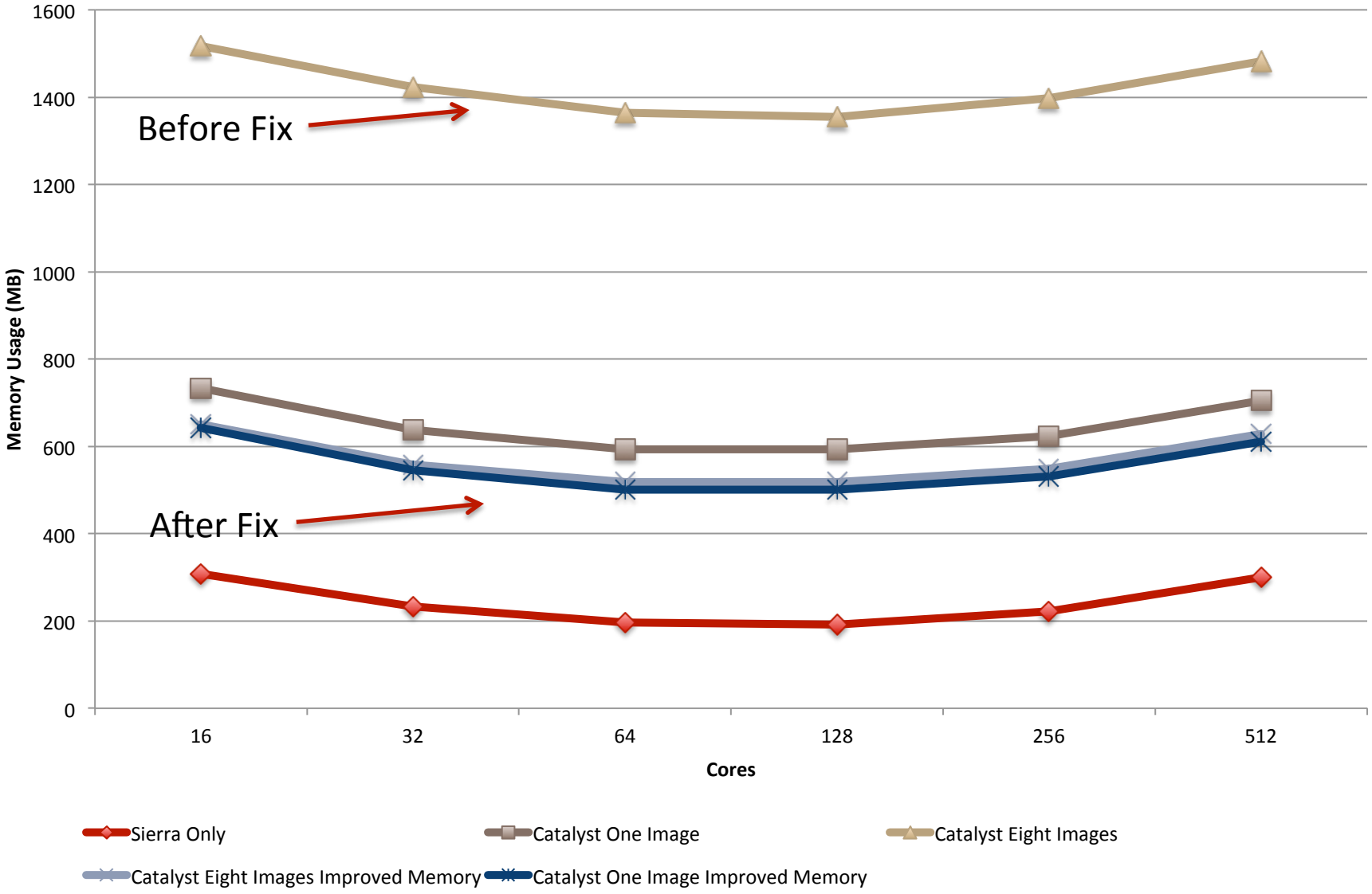
Sandia National Laboratories

Per Core Max Total Memory in Usage
(non-heap memory shared among 8 cores per node)



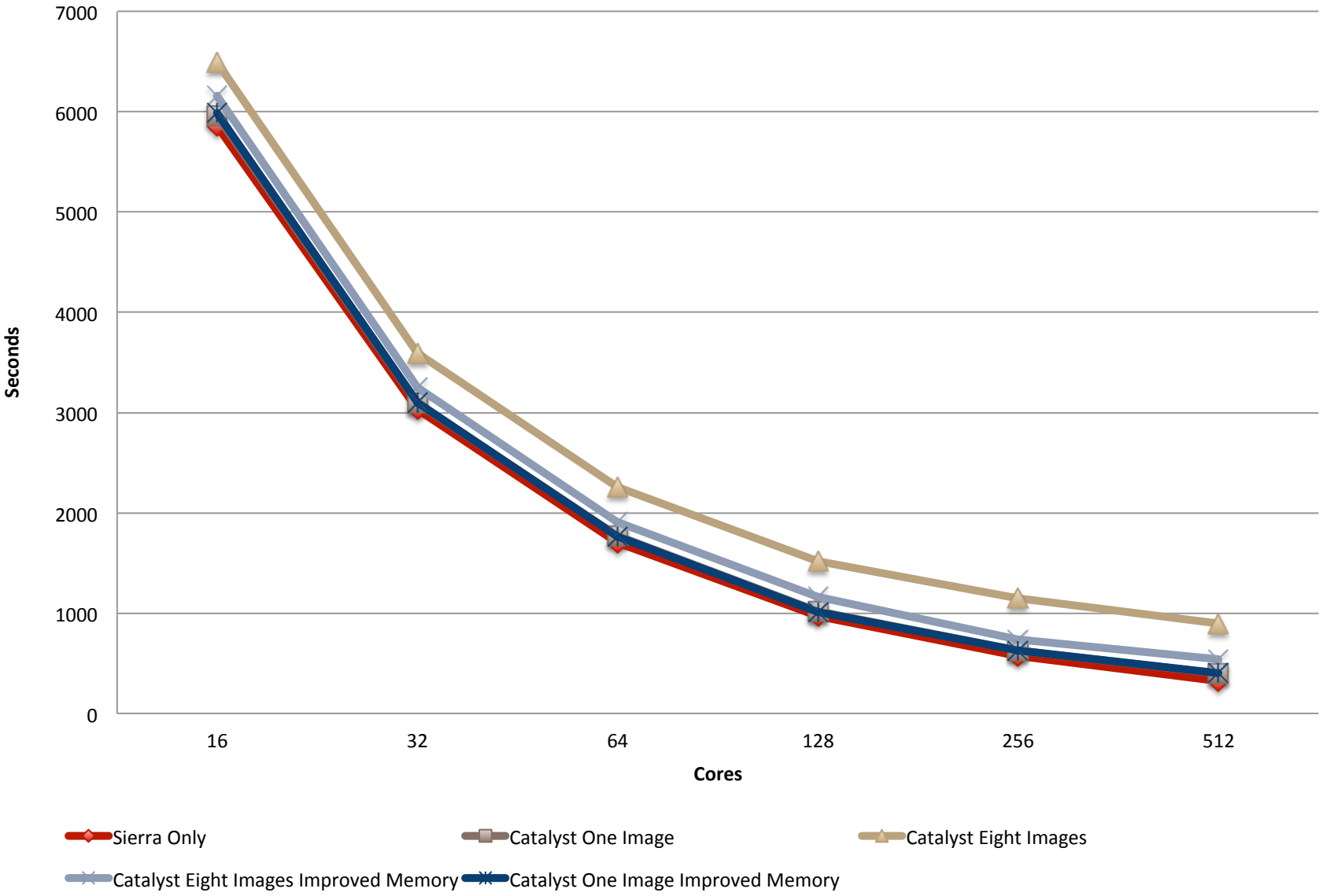
Sandia National Laboratories

Per Core Max Heap Allocation



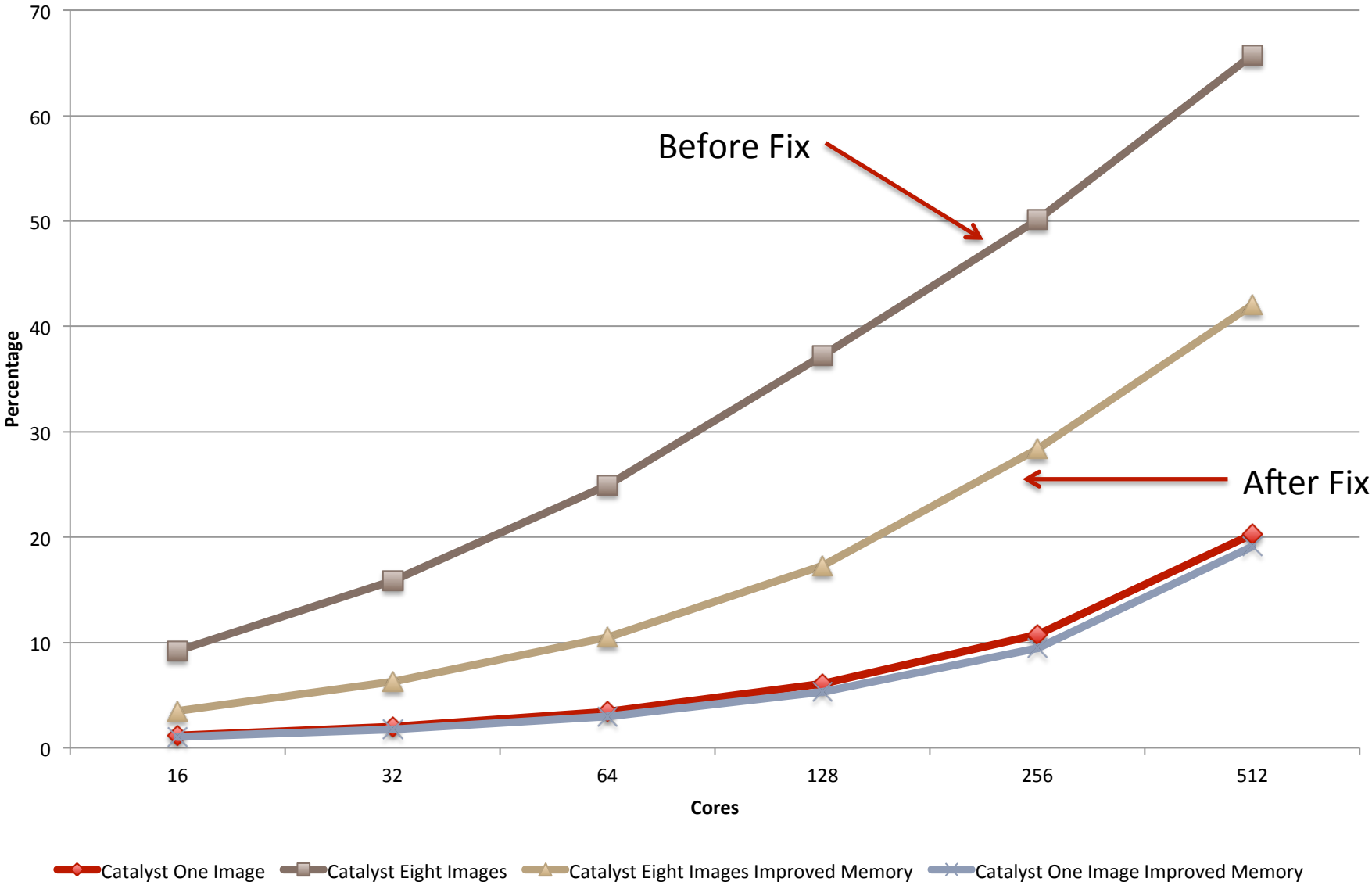
Sandia National Laboratories

Total Execution Time



Sandia National Laboratories

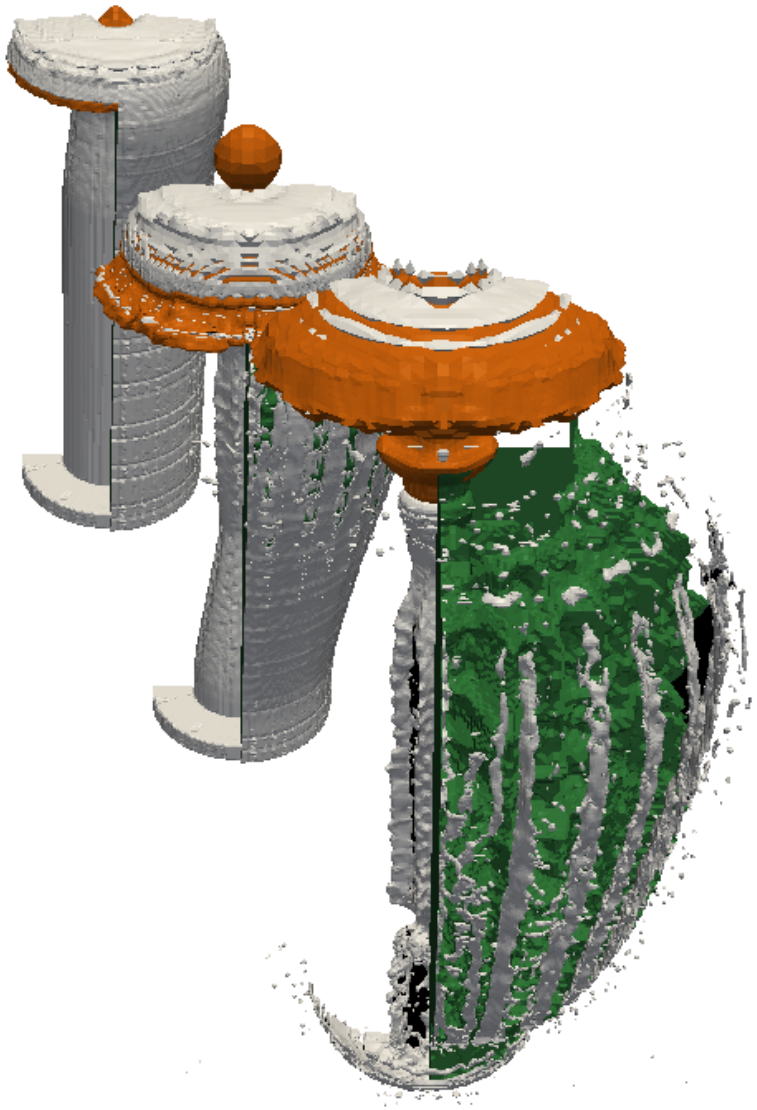
Percentage Of Total Execution Time Spent In Catalyst



Experiment Driver (SAND2013-1122)

Sandia National Laboratories

- Customer use case: characterize fragments in a shock physics explosive simulation
- Code: CTH
- Analyst: Jason Wilke
- Critical steps
  - Find fragments (multiple operations required)
  - Characterize fragments (mass, velocity, etc.)
  - Extract useful information



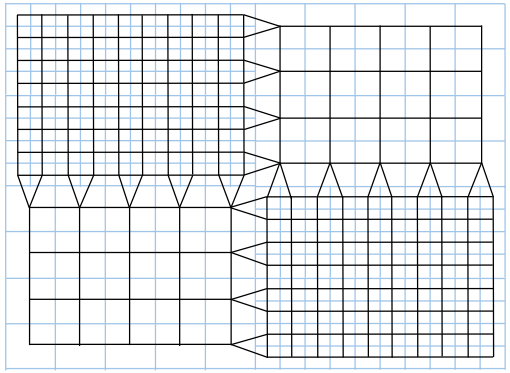
Identifying fragments is a complex part of the analysis, and serves as a useful way to characterize the operations.

The full range of data experiments was run at 32k cores on Cielo. Partial experiments were performed at 64k cores. This report presents results from the 32k core runs.

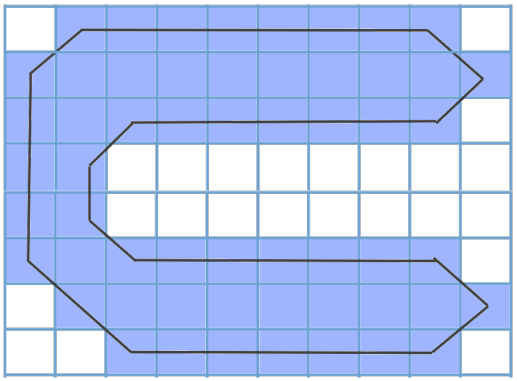
Fragment detection

Sandia National Laboratories

- Operations required for fragment detection (requires a watertight surface)
  1. Find block neighbors
  2. Build a conforming mesh over AMR boundaries
  3. Identify boundaries of fragments



Step 1 & 2



Step 3

Implemented Workflows

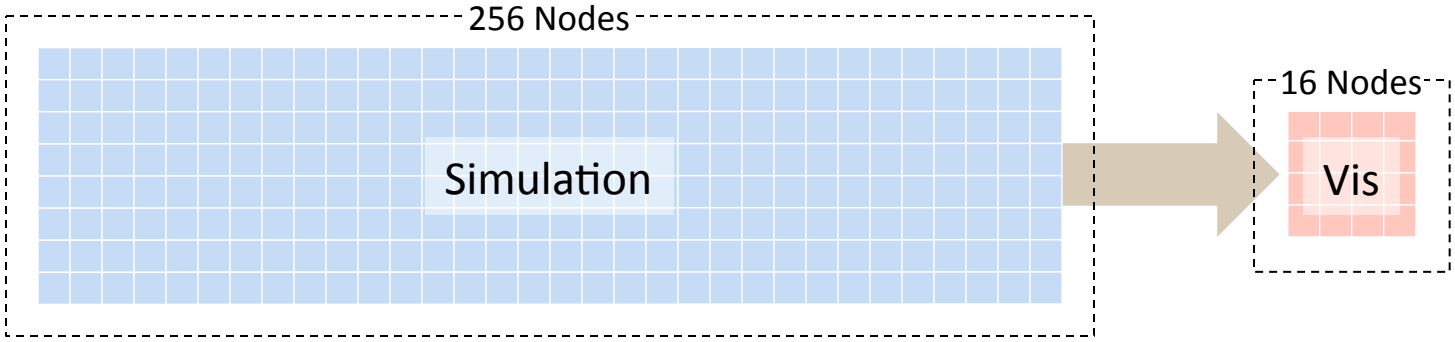
Sandia National Laboratories

- ***In situ***: A CTH job that directly runs *in situ* data analysis
  - **Baseline**: Basic algorithm with somewhat redundant step of global communication to find AMR block neighbors
  - **Refined**: Improved algorithm that gets AMR block neighbors from CTH
- ***In transit***: CTH transfers data to separate server job
  - **Extra nodes**: CTH job size same as other runs, extra nodes are used to allocate the VDA service
  - **Internal nodes**: CTH job given fewer nodes that are assigned to VDA service so that together both jobs use the same nodes as other runs
- **Post-processing**: Write Spyplot files from CTH, then post process with ParaView batch script

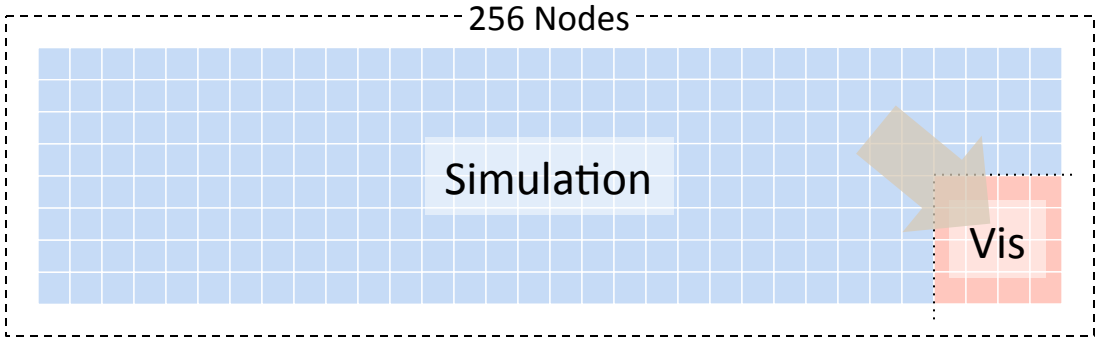
In Transit Allocations

Sandia National Laboratories

"Extra Nodes" allocated for VDA services



"Internal Nodes" included in job allocation



Experiment Configurations

Sandia National Laboratories

- All experiments performed on Cielo supercomputer at LANL, jointly managed by Los Alamos National Laboratory and Sandia National Laboratories
  - 8,944 node Cray XE6
  - Node: 2 AMD Opteron 6136 (Magny-Cours) 8-way processor chips
    - Total of 16 cores/node
    - 2.4 GHz peak computation speed per core
  - Peak of 1.37 Petaflops
  - 32 GB memory/node

Experiment, cont'd

Sandia National Laboratories

- All experiments completed 500 cycles (i.e., timestep calculations) of the CTH code.
- The first four experiments executed an analysis operation once every 10 cycles
- For standard full-mesh data output, the CTH code was set to output the same number of time steps as the in situ and in transit experiments
  - Total number of analysis operations is the same
- Data captured was from instrumented code and HPCToolkit

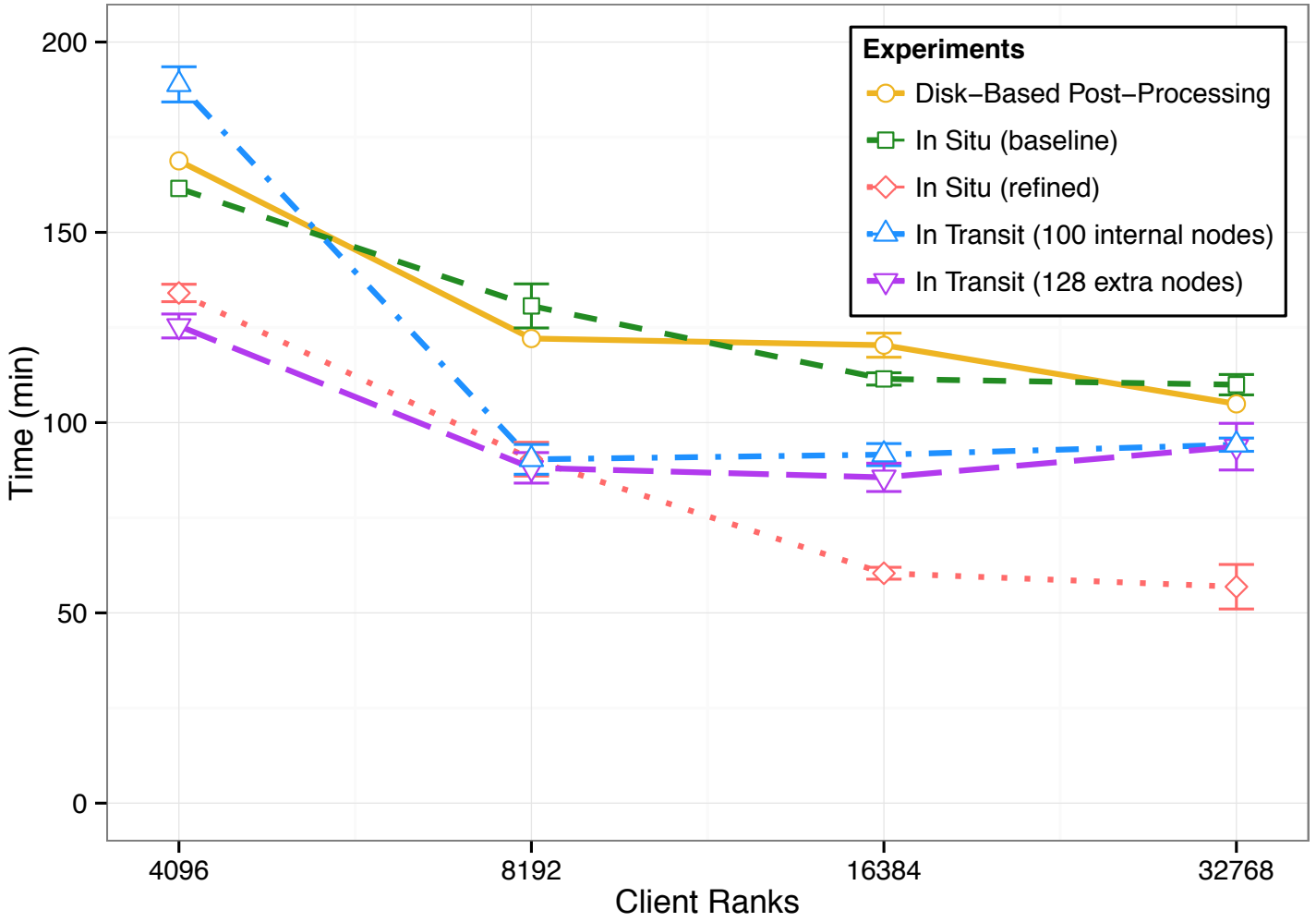
Experiment, cont'd

- Each experiment was run in a strong scaling fashion with three different datasets.
  - Each data set comes from the same initial conditions but with a different maximum level of refinement
  - Measurements of different job sizes with different data set sizes provides a weak scaling overview.

CTH				*In transit* Server			
Most		*In transit* Internal		Extra Nodes		Internal Nodes	
Cores	Nodes	Cores	Nodes	Cores	Nodes	Cores	Nodes
33K Blocks — 5 levels							
128	8	96	6	16	2	16	2
256	16	224	14	16	2	16	2
512	32	480	30	16	2	16	2
1,024	64	992	62	16	2	16	2
220K Blocks — 6 levels							
1,024	64	768	48	128	16	128	16
2,048	128	1,792	112	128	16	128	16
4,096	256	3,840	240	128	16	128	16
8,192	512	7,936	496	128	16	128	16
1.5M Blocks — 7 levels							
4,096	256	2,496	156	1,024	128	800	100
8,192	512	6,592	412	1,024	128	800	100
16,384	1,024	14,784	924	1,024	128	800	100
32,768	2,048	31,168	1,948	1,024	128	800	100
65,536	4,096	63,936	3,996	1,024	128	800	100

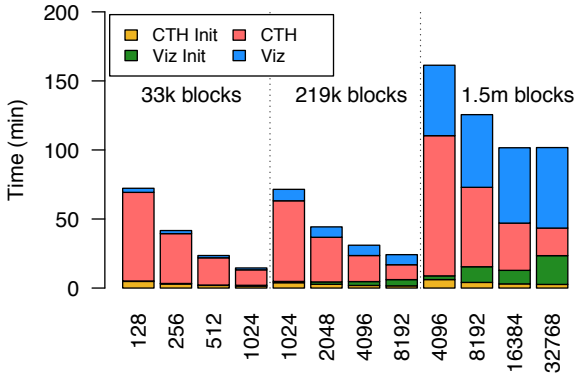
Pipeline Summary Timing (1.5m blocks)

Sandia National Laboratories

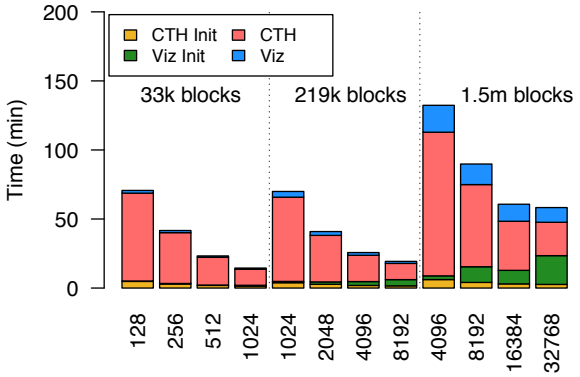


Acceptable scaling performance, with the exception of the baseline algorithm.

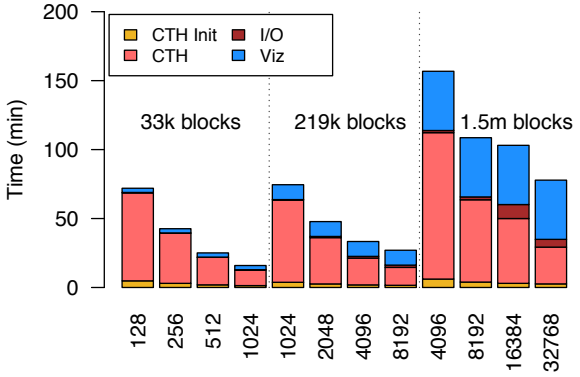
Timing Per Task



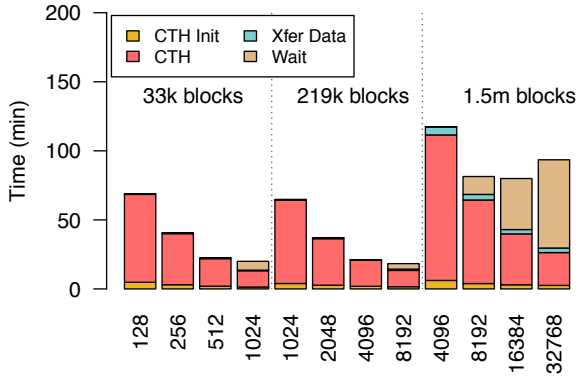
In situ baseline



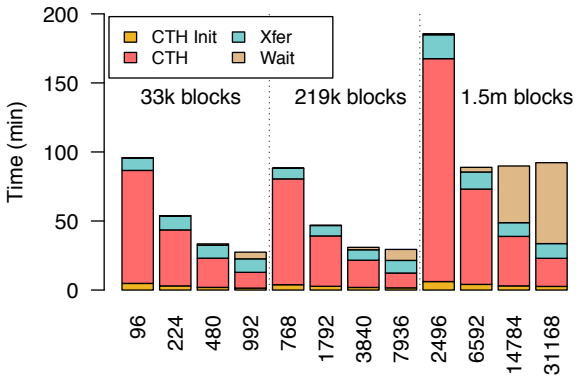
In situ refined



Disk-based post-processing



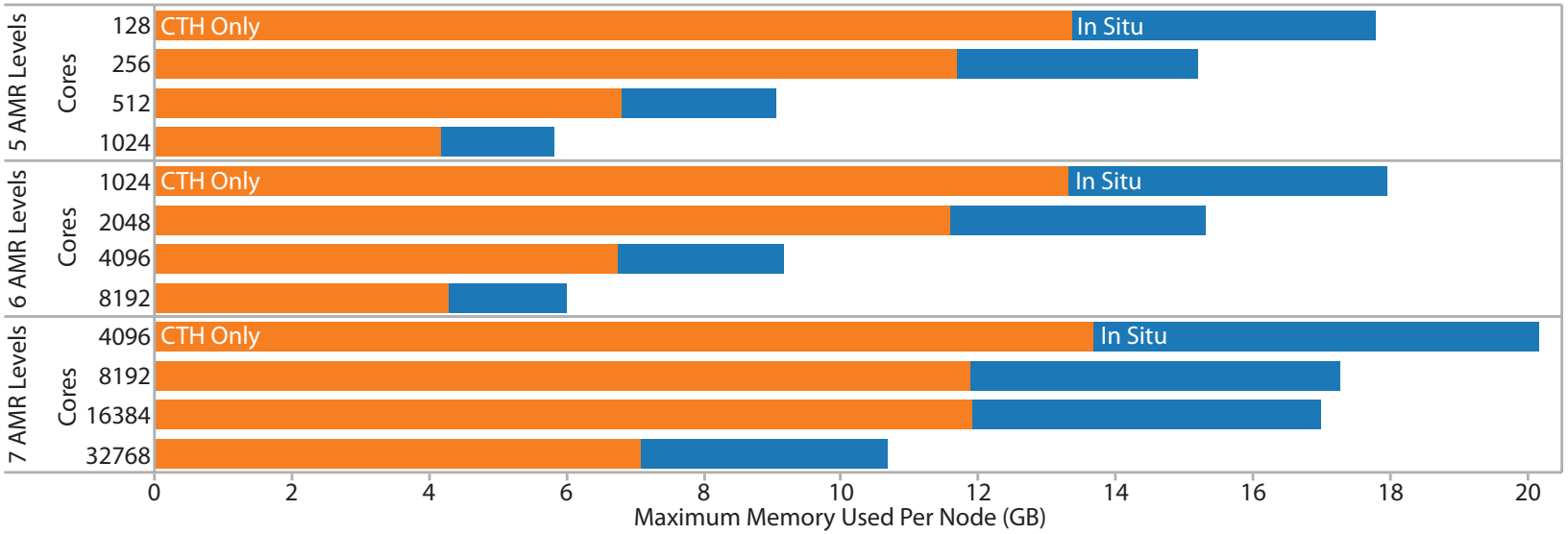
In transit extra nodes

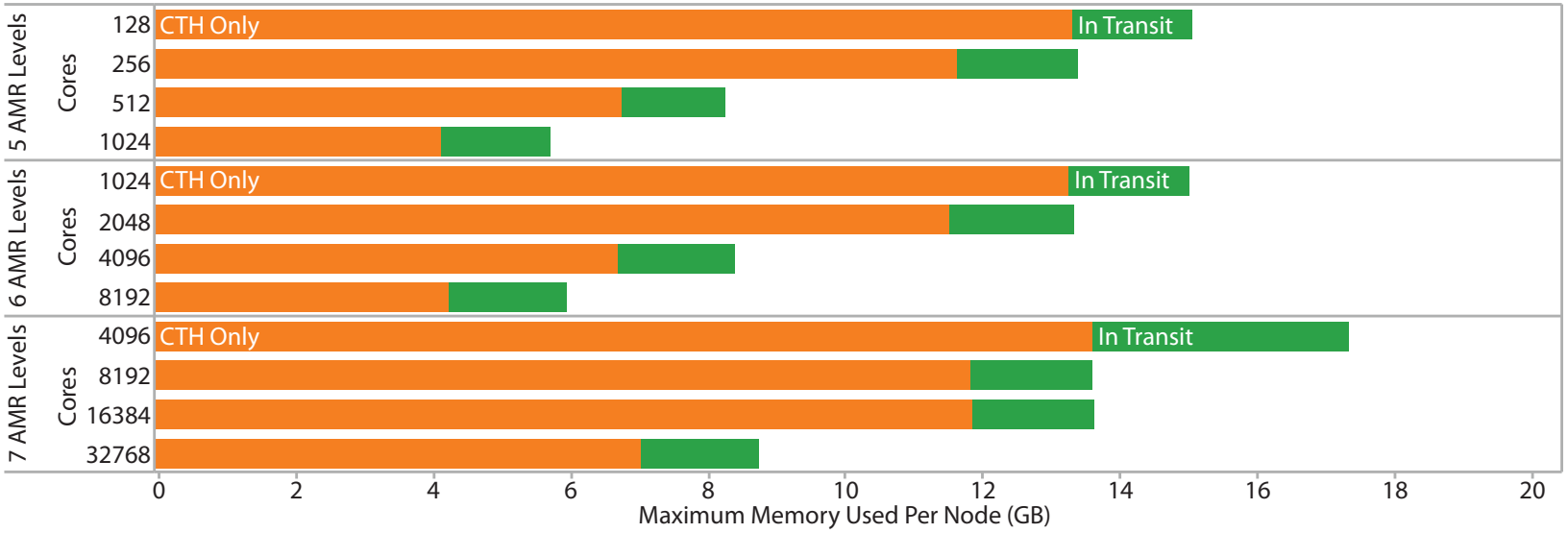


In transit internal nodes

- CTH scales well.
- Baseline algorithm does not scale
- Disk I/O not bad

Memory Footprint (on code side)





Memory overhead generally falls between 25% and 50%